adapya-base Documentation

Release 1.0.4

software AG

Dec 03, 2019

Contents

1 Overview **3**
- 1.1 adapya-base License . . . 3
- 1.2 Change History . . . 3

2 Installation **5**
- 2.1 Prerequisites . . . 5
- 2.2 Installation with Package Installer . . . 5
- 2.3 PYTHONPATH Installation . . . 6
- 2.4 Additional Windows Installation Notes . . . 7
- 2.5 Unix/Linux PYTHONPATH Installation . . . 7

3 Using adapya-base **9**
- 3.1 Read/Write Buffers with Abuf . . . 9
- 3.2 Working with Structures with Datamap . . . 10

4 Scripts **13**
- 4.1 ftpz.py - synchronize PDS with local files in folders . . . 13
- 4.2 getfilez.py - transfer single file and optionally dump records . . . 15
- 4.3 smfreaderz.py - read and print SMF30 records . . . 16

5 Package Reference **19**

6 Indices **21**

adapya-base 1.0.4

CHAPTER 1

Overview

adapya-base provides basic functions used by other adapya packages like e.g. adapya-adabas. It comes with sample programs and scripts to show its use.

adapya-base is a **pure** Python package: it does not require compilation of extensions, but uses the **ctypes** module.

It has been used on mainframe z/OS, Solaris, z/Linux and Windows.

Further information on adapya can be found at

http://tech.forums.softwareag.com/techjforum/forums/show/171.page

1.1 adapya-base License

1.2 Change History

adapya-base 1.0.0 (June 2018)

- jconfig uses keyring package for passwords if installed
- scripts for z/OS file transfer and processing SMF records

adapya-base 0.9.5 (April 2017)

- move adapya-base functions into separate defs module
- support of z/OS with the Rocket Python 2.7 and 3.6

adapya-base 0.9 (September 2016)

- adapya was split into smaller packages to achieve independence
- support of Python 3.5 and higher
- support of z/OS with the Rocket Python 2.7.12

Adapya 0.8

- dtconv.py new routine for date/time conversions
- Datamap added support for
  - multiple fields and periodic groups
  - packed and unpacked format
  - mapping datetime() objects to DATETIME, TIMESTAMP U fields

adapya 0.7 is the first public release.

CHAPTER 2

Installation

2.1 Prerequisites

Before installing adapya ensure the following:

- Python is available on the platform.

  adapya supports the Python versions 2.7 or 3.5 and higher

  If you have no Python on the system you may install Python V2.7 or V3.6. The Python installer can be downloaded from

  http://www.python.org/download/

- the **ctypes** package has been ported (usually installed with Python on Windows, Linux and Rocket Python on z/OS from version 2.7.12)

  On some platforms this may not be available because of issues with the underlying forreign function package that does not support the hardware.

- Sudo rights simplify standard installation (Unix/Linux).

Note: Users starting with Python are advised to read the Python Tutorial available with function key F1 in the IDLE Python GUI or at https://docs.python.org/3/tutorial/index.html

2.2 Installation with Package Installer

The Python package installer offers the most convenient method of installing packages.

From Python package index web site:

```
> pip install adapya-base
```

From zip file (similarly for tar file):

> pip install adapya-base-1.0.4.zip

2.2.1 Additional z/OS Installation Notes

Rocket Software has released Python versions 2.7 and 3.6 for z/OS to run unter Unix System Services.

For installing the package under one of those versions perform the following steps:

- FTP transfer the zip file in binary mode
- execute pip with the following parameters (in one line):

  pip install -U –no-index –disable-pip-version-check –no-binary all adapya-base-1.0.4.zip

2.3 PYTHONPATH Installation

Alternatively, the PYTHONPATH installation allows for temporary package installation by adding the location of the package to the PYTHONPATH environment variable. As location may serve the directory where the package was extracted to or the package zip file itself.

When the Python interpreter is started it evaluates the environment variable **PYTHONPATH** and adds any directories listed to its search path for modules.

For example, on Windows the following steps would do a PYTHONPATH installation:

- The zip file adapya-base-1.0.4.zip contains a directory adapya/base/*
- Unzip adapya-base-1.0.4.zip to a convenient location e.g.:

```
> C:/ADA/Python
```

  maintaining the subdirectory structure

- Set/check the following system variables

  On Windows (Win-key + PAUSE-key) open the System Control / select Extended Control / button Environment Variable:

```
> REM adapya-base Python directory
> set PYB=C:\ada\python\adapya-base-1.0.4
> set PYTHONPATH=%PYB%;%PYTHONPATH%
```

- Open a cmd window and go to Adabas demo files directory:

```
> cd %PYB%/adapya/base
```

- Check successful installation with dump.py to dump contents of file:

```
> python dump.py -f future.py
```

2.4 Additional Windows Installation Notes

2.4.1 Simplifying Execution of Python Scripts

The option to register Python files can be selected during the Python installation. This binds certain Python file types and associations to the Python executable being installed (or to the Python launcher py.exe).

For example for .py the following may have been set:

```
ftype Python.File="C:\Windows\py.exe" "%L" %*
ftype Python.ArchiveFile="C:\Windows\py.exe" "%L" %*
ftype Python.CompiledFile="C:\Windows\py.exe" "%L" %*
ftype Python.NoConArchiveFile="C:\Windows\pyw.exe" "%L" %*
ftype Python.NoConFile="C:\Windows\pyw.exe" "%L" %*

assoc .py=Python.File
assoc .pyc=Python.CompiledFile
assoc .pyo=Python.CompiledFile
assoc .pyw=Python.NoConFile
assoc .pyz=Python.ArchiveFile
assoc .pyzw=Python.NoConArchiveFile
```

If you add **.py** and the corresponding compiled extensions to the PATHEXT variable it is possible to run a script without writing the extension

```
set PATHEXT=.py;.pyc;.pyo;%PATHEXT%
dump -h
```

rather than typing:

```
dump.py -h
```

2.5 Unix/Linux PYTHONPATH Installation

The PYTHONPATH environment variable defines an extra search path for python modules. If the path to the Adabas Python directory is added to the variable it is included in the search:

```
cd /FS/disk01/pya            # root directory
tar xf adapya-base-1.0.4.tar # unpack to adapya-base-v.r.l
setenv PYA "/FS/disk01/pya"
setenv PYTHONPATH $PYA':'$PYTHONPATH # add PYA to PYTHONPATH
cd $PYA/adapya/base # go to directory
python dump.py -h
```

Note: If your local internet is protected by a http proxy you may need to set the HTTP_PROXY environment variable before running easy_install (CYGWIN):

```
SET HTTP_PROXY=http://<httpprox.your-local.net>:<httpprox-port>
```

Not setting it may result in timing out operations.

CHAPTER 3

Using adapya-base

3.1 Read/Write Buffers with Abuf

Implemented in *adapya.base.defs*

Read/write buffers are not directly available in Python only indirectly with I/O routines.

The adapya.base.defs.Abuf class implements read/write buffers. They are used for the Adabas control block and other buffers.:

```
>>> from adapya.base.defs import Abuf # Access Abuf class
>>> a=Abuf(8) # Allocate buffer of 8 bytes
>>> a.value=b'Bell' # store 'Bell' into buffer a
>>> a.raw # show contents of a
b'bell\x00\x00\x00\x00'

>>> a[0:8]   # same in slice notation
b'bell\x00\x00\x00\x00'

>>> a[0:4]  # extract part of buffer
b'bell'

>>> a[0]=b'W' # modify buffer at offset 0
b'Well\x00\x00\x00\x00'

>>> a.read(5) # read() 5 bytes of the buffer
b'Well\x00'

>>> a.read(5) # read() next bytes (only 3 left)
b'\x00\x00\x00'

>>> a.tell() # inquire position
8

>>> a.seek(0) # position to start of buffer
```

(continues on next page)

(continued from previous page)

```
>>> a.write('Sun') # write first 3 characters
b'Sunl\x00\x00\x00\x00'
```

3.2 Working with Structures with Datamap

Implemented in *adapya.base.datamap*

The datamap module defines the Datamap class that allows to define structure within a byte buffer. This is similar to a DSECT or C struct. This is being used for setting up the Adabas control block and the other buffers like record buffer or value buffer.

3.2.1 Datamap Data Types

The following data types require a name and a length:

String alpha numeric string

Unicode Unicode string (in UTF-16, each character takes 2 bytes)

Bytes similar to String but hexadecimal contents

Packed integer mapped to P format

Unpacked integer mapped to U format

The other data types take only the name:

Char 1 byte character

Int1/2/4/8 signed integer of 1, 2, 4 or 8 bytes

Uint1/2/4/8 unsigned integer of 1, 2, 4 or 8 bytes

Float single precision float (IEEE format)

Double double precision float (IEEE format)

3.2.2 Basic Usage example

```
>>> from adapya.base.defs import Abuf
>>> from adapya.base.datamap import Datamap, String, Int2
>>> dm=Datamap('mymap',
        String('name',6),  # List of field definitions. Its order
        Int2('age')) # defines the position in the buffer
>>> dm.getsize() # query the size of the datamap dm
8
>>> a=Abuf(8)
>>> dm.buffer=a  # use buffer a with datamap dm
>>> dm.name='Bell' # assign name
>>> dm.age=64 # assign age
>>> dm.dprint()  # print out all values of datamap dm
mymap
name = "Bell"
age = 64
```

(continues on next page)

(continued from previous page)

```
>>> dm.age  # individual attribute access
64
>>> dm.name
'Bell'
>>> dm.name='12345678' # silent truncation (field size defined = 6)
>>> dm.name
'123456'
>>> dm.name='1234 ' # silent blank truncation on return
>>> dm.name
'1234'
```

3.2.3 Example showing the different data types

```
from adapya.base.datamap import Datamap, String, Unicode, Utf8, Bytes,
    Char, Int1, Uint1, Int2, Uint2, Int4, Uint4, Int8, Uint8,
    Float, Double, Unpacked

p = Datamap( 'test_all_formats',
       String( 'str8', 8),
       Unicode('uni4', 4), # unicode 4 chars = 8 bytes
       Utf8( 'utf8', 8),
       Bytes( 'byt4', 4),
       Char( 'cha1'),
       Int1( 'int1'),
       Uint1( 'uin1'),
       Int2( 'int2'),
       Uint2( 'uin2'),
       Int4( 'int4'),
       Uint4( 'uin4', opt=T_STCK), # Uint4 STCK
       Int8( 'int8'),
       Uint8( 'uin8', opt=T_STCK), # Uint8 STCK
       Float( 'flo4'),
       Double( 'dou8'),
       Unpacked( 'dati', 14, dt='DATETIME'), # Python datetime object
       )
```

With Uint4 and Uint8 the additional option T_STCK indicates that the value is a timestamp value in mainframe STCK format. This is evaluated with the dprint() function to print the timestamp in readable ISO format.

An *Unpacked* field defined with the dt option (value 'DATETIME' or 'TIMESTAMP') work with Python datetime() objects:

```
>>> from datetime import datetime
>>> p.dati = datetime.now()
>>> print p.dati
2010-05-01 11:55:00

>>> _,pos,ln,_,_ = p.keydict['dati']
>>> p.buffer[pos:pos+ln]
'20100501115500'
```

Note that a datetime value of zero corresponds to the Python **None** type:

```
>>> p.dati = None
>>> print p.dati
```

(continues on next page)

(continued from previous page)

```
None
>>> p.buffer[pos:pos+ln]
'00000000000000'
```

3.2.4 Defining Multiple fields

If the keyword **occurs** is added to the end of the field definition it becomes a multiple value field.

Note: The field access does not return the field value but an iterator over the occurrences values - see m.languages as in the example below:

```
>>> from adabas.datamap import Datamap, String
>>> from adabas import Abuf
>>> m = Datamap('mulitple_demo', String('languages', 3, occurs=5),
            buffer=Abuf(100))
>>> m.languages=('ENG','FRA','GER')
>>> m.languages[2]
'GER'
>>> m.languages[2]='RUS'

>>> for lang in m.languages:
      print lang
ENG
FRA
RUS
```

3.2.5 Defining Periodic Group

If the field definition within a datamap contains a periodic () definition a datamap at the present location is defined.

```
from adapya.base.datamap import Datamap, String, Packed
p = Datamap( 'Periodics Demo',
             ...
             Periodic(
               Datamap('Income',
                 String( 'currency', 3),
                 Packed( 'annual', 5)
                 ),
               occurs=8),
             ...
             )

>>> print len(p.income)
8

p.income[0].currency = 'EUR'
p.income[0].annual = 99000
```

CHAPTER 4

Scripts

There are some scripts in adapya.base.scripts that can be run on the command line. Usually they accept Unix style parameters. A help page is shown with the help option.

- ftpz.py - synchronize partitioned datasets with local folders
- getfilez.py - transfer single file and dump records
- smfreaderz.py - read and interpret SMF30 records

4.1 ftpz.py - synchronize PDS with local files in folders

ftpz.py synchronizes remote partitioned datasets on z/OS with local files in folders.

The syncronization depends on the modification time stamps in the file system and the PDS directory

Usage: ftpz.py [options]:

```
[-h] [-k] [-e [EXT]] [-n HOST] [-p PWD] [-s SITE] [-u USER]
[-r] [-t] [-v VERBOSE] [-d PREFIX] [-x [EXCLUDE [EXCLUDE ...]]]
[-i [INCLUDE [INCLUDE ...]]]
{upsync,upload,download,downsync,config} [pds]
```

Positional arguments:

{upsync,upload,download,downsync,config}:

- **upsync: upload changed members to partioned dataset (PDS) and** delete members not existing in source
- upload: upload changed files to partioned datasets (PDS)
- download: download changed files from partioned datasets (PDS)
- **downsync: download changed members from partioned dataset (PDS)** and delete members not existing in PDS

- **config: display or set configuration pds -** Partitioned Dataset name w/o quotes. With ending period: parameter is used as high-level qualifier

Optional arguments:

```
-h, --help            show this help message and exit
-k, --keepsame        On download/sync do not overwrite old file if contents
                      same as new file
-e [EXT], --ext [EXT]
                      extension to process (default is .s) e.g. -e .c -e
                      without argument: no extension
-n HOST, --host HOST  ftp z/OS host name
-p PWD, --pwd PWD     ftp z/OS user id
-s SITE, --site SITE  Set ftp server options per quote site
-u USER, --user USER  ftp z/OS user password
-r, --ignerr          ignore ftp member access errors
-t, --test            dry run - list actions only
-v VERBOSE, --verbose VERBOSE
                      verbose output (1 to 3)
-d PREFIX, --prefix PREFIX
                      dataset prefix
-x [EXCLUDE [EXCLUDE ...]], --exclude [EXCLUDE [EXCLUDE ...]]
                      list of libraries/members to exlude from processing
-i [INCLUDE [INCLUDE ...]], --include [INCLUDE [INCLUDE ...]]
                      list of libraries/members to include in processing
```

Examples:

1. sync-up PDS mm.test.adasrc from current directory and delete member that do not exist in source:

```
> ftpz upsync --user xyz --pwd secret --host zmax MM.TEST.ADASRC
```

2. update all changed members PDS mm.test.adasrc from current directory:

```
> ftpz upload --user xyz --pwd secret --host zmax MM.TEST.ADASRC
```

3. ftpz upsync -t mm.test.adasrc # dry run without updating

4. set configuration default values:

```
> ftpz config --user xyz --pwd secret --host zmax # set config
> ftpz config                                     # list config

> ftpz upload MM.TEST.ADASRC                      # use config
```

When a configuration file is set up with the most common parameters in use - (user, host and password (pwd)) these do not have to be typed on each script execution. The password is encrypted so that it is not plainly readable on the command line. See example 2.

5. download

   Download all missing or changed members of PDS mm.test.adasrc to current directory:

```
> ftpz download MM.TEST.ADASRC
```

6. downsync

   Same as 'download' and in addition delete members in directory that are missing in PDS:

```
> ftpz downsync MM.TEST.ADASRC
```

7. download all changed members for all PDS starting with high-level quailfier (HLQ).

   The current directory is used as the base matching its subdirectories name with partitioned datasets HLQ.name. Optionally exclude or include parameters can specifiy a list of libraries to exclude or include separated by space:

   ```
   > ftpz download mm.test. --exclude unwanted abandond
   ```

   will process pds mm.test.source and mm.test.macros but exclude mm.test.unwanted and mm.test.abandond

8. download specific missing or changed members 'FOO' and 'BAR' of a PDS as foo.s and bar.s in current directory:

   ```
   > ftpz download mm.test.source --include foo bar
   ```

9. download to use extension .c and special translation table for ibm-1047 to ibm-819 with EBCDIC LF = 0x15

   > ftpz download mm.c.source -e .c -s sbdataconn=MM.OEMVS31.TCPXLBIN

4.2 getfilez.py - transfer single file and optionally dump records

Read specific dataset or PDS member from z/OS per FTP converted to ASCII or binary.

Datasets may be variable blocked sequential dataset as binary with RDW record prefix.

Usage: getfilez [options]

The records of the local file can be dumped setting the –verbose switch 4 and a selected with –numrec and –skiprec parameters (example 3 below).

Options:

```
-a  --ascii         transfer with EBCDIC to ASCII conversion
-b  --binary        binary transfer (variable blocked) with RDW prefix
-d  --dsn           remote sequential dataset name
-e  --ext           extension (default .s) for member names if no
                    fname specified
-f  --fname         local file name (optional)
-c, --config        set/show configuration

-n  --numrec        with verbose & 4: number of records to print
-p, --pwd           <password>  FTP ser1.0.4ogin password (*)
-u, --user          <userid>                              (*)

-r, --recform       specifies the record structure:
                    'RDW' variable records inlcude Record
                          Descriptor Word which is skipped
                    'RDW+' same as RDW but also return RDW
                    'BDW' data includes Block Descriptor Word
                          which is skipped (RECFM=U)
                    'BDW+' same as BDW, bu also return record with RDW
                    'EXCL4' 4 byte excl. length prefix
-s  --skiprec       with verbose & 4: number of records to skip
-v, --verbose       0: (default), 1: log ftp, 2: detailed ftp,
                    4: dump records
-x, --xlate         full dataset name of the hlq.name.TCPXLBIN translate
```

(continues on next page)

(continued from previous page)

```
                    table on mainframe for EBCDIC to ASCII conversion
                    using the "site SBDATACON=<xlate>
-h, --host          <host name> of IBM FTP server         (*)
-?, --help
```

Defaults marked with (*) are taken from configuration (-c) The configuration values are stored ciphered in file ~/.toolz

Examples:

1. set configuration user, password:

```
> getfilez --config --user hugo --pwd secret
```

2. read remote dataset with verbose FTP operations, user and password are taken from configuration. File is processed binary and RDW record headers are preserved:

   > getfilez -bd mm.db8.uld1 -r RDW -h da3f -v2

3. dump VB records in local file limited by skiprec and numrec:

```
> getfilez -f mm.db8.uld1 -r RDW -v4 -n 1000 -s 1222000
```

4. copy member EPILOG from PDS to local file epilog.s and convert to ASCII:

```
> getfilez -ad mm.pds(epilog)
```

4.3 smfreaderz.py - read and print SMF30 records

Usage: smfreaderz [options]:

```
options:

    -d  --dsn      <smf dataset name>  remote SMF file
    -f  --file     <file> local SMF file
    -b  --bfile    <file> local SMF file VB blocked with BDW

    -k, --skiprec  <int>   number of records to skip
    -m, --maxrec   <int>  maximum number of records
    -p, --pwd      <password>  FTP ser1.0.4ogin password (*)
    -u, --user     <userid> FTP ser1.0.4ogin userid      (*)
    -h, --host     <host name> of IBM FTP server         (*)

    -s, --select   <record selection criteria> see below (**)

    -c, --config   Set/show configuration
    -v, --verbose  level of printed information (default 2)
    -?, --help

(**) record selection criteria  kw1=val1[,kw2=val2]
     enclose hole string with " if it contains blanks
     valid keywords: job, id, user, group, prog
     Criteria must be all fulfilled to select a record
     Example: -s job=*MM*,group=RND,id=J*,prog=*ASM

(***) verbose level, composable: 1 - stats SMF records
```

(continues on next page)

(continued from previous page)

```
        2 - detailed print selected SMF records, 4 - dump records,
        8 - debug


Defaults marked with (*) are taken from configuration.
The configuration for user specific parameters can be stored
with the --config option.

The reader can transfer the file (--dsn) per FTP from a remote z/OS
with the RDW option or can access the file locally if already
transfered (--file). On z/OS the --bfile option may be used.

Option -b/--bfile if file includes block descriptor word (BDW)
e.g. when running on z/OS with DCB=(RECFM=U) override on DD stmt


Examples:

1. set configuration user, password
    smfreaderz --config --user hugo --pwd secret

2. read remote SMF dataset and print
    smfreaderz -d cc.sysa.smf -h sysa
```

The following command (on Windows cmd) will select SMF30 records with program name ADARUN from a SMF system dataset:

```
>>> smfreaderz -v3 -s JOB=MM10026 -d ZMAX.SMFDAY.G3037V00
```

Record selected by condition [' JOB=MM10026']:

```
--- Record 181197: SMF30 ---

Product or Subsystem Section
SMF30 sub type         = Step total
Record version number  = '05'
Subsystem product name = 'SMF'
MVS product level      = 'SP7.2.1'
System name            = 'ZMAX'
Sysplex name           = 'MAXPLEX'


Job/Session Id Section
Job/session name             = 'MM10026'
Program name                 = 'ADARUN'
Step name                    = 'ADANUC'
JES job id                   = 'S0207297'
Step number                  = 1
Device allocation start time = 14:35:54.95
Problem program start time   = 14:35:55.23
Time initiator selected step = 14:35:54.95
Date initiator selected step = 2018.079
Time reader found job card   = 14:35:54.79
Date reader found job card   = 2018.079
Time reader found end of job = 14:35:54.82
Date reader found end of job = 2018.079
```

(continues on next page)

(continued from previous page)

```
RACF group id                = 'MFRAME'
RACF user id                 = 'RACFSTC'
Step name invoking procedure = 'STARTING'
Job class                    = 'STC'
Interval start time          = 2018-03-20 14:35:54.955237.078
Interval end time            = 2018-03-20 14:56:26.669574.079
Address space id             = X'012E'


CPU accounting section
Timer Flag1                                  = X'80'
Step CPU time under TCB                      = 00:00:45.18
Step CPU time under SRB                      = 00:00:12.24
Initiator CPU time under TCB                 = 00:00:00.30
CPU time I/O Interrupts                      = 00:00:06.26
Step dependent enclave CPU time              = 00:00:44.62
Time on zIIP                                 = 00:00:42.66
Dependent enclave time on zIIP               = 00:00:42.66
zIIP time on CP                              = 00:00:03.06
Dependent enclave zIIP time on CP            = 00:00:03.06
Dependent enclave zIIP time on CP normalized = 00:07:51.38
CPU TCB time for step init                   = 00:00:00.24
Highest Task Program name                    = 'IEESB605'

Performance section
zIIP normalization factor = 1.98
```

To run this in z/OS batch the dataset must be referenced via DD name 'SMF':

```
//MMSMF30  JOB MM,CLASS=G,MSGCLASS=X,LINES=100
//*
//* smfreaderz.py reads SMF files from DD:SMF
//* -h option will print usage / description
//*
//BPX      EXEC PGM=BPXBATSL
//SMF DD DISP=SHR,DSN=OPS.ZMAX.SMFDAY.G3037V00,DCB=(RECFM=U)
//STDPARM  DD *
PGM /usr/mm/py27/bin/python
   /usr/mm/apy/smfreaderz.py -b dd:SMF -v3
   -s JOB=MM10026
/*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD PATH='/usr/mm/apy/batsl.env',PATHOPTS=ORDONLY
//
```

CHAPTER 5

Package Reference

CHAPTER 6

Indices

- genindex
- modindex